

# Concur : An Algorithm for Merging Concurrent Changes without Conflicts

James Smith  
jecs@imperial.ac.uk

## Abstract

Suppose you and I are both editing a document. You make a change and I make a change, concurrently. Now if we want to still be seeing the same document then I need to apply your change after mine and you mine after yours. But we can't just apply them willy-nilly. I must amend yours somehow and you mine. If my change is written  $\Delta$ , yours  $\delta$ , my amended change  $\delta.\Delta$  and yours  $\Delta.\delta$ , we get  $*\Delta * \Delta.\delta = *\delta * \delta.\Delta$  as long as application is written  $*$  and we don't care about what we're applying the changes to. We start by proving this identity for single changes and finish by proving it for many.

*As I was eating, I saw Salvatore in one corner, obviously having made his peace with the cook, for he was merrily devouring a mutton pie. He ate as if he had never eaten before in his life, not letting even a crumb fall...He winked at me and said, in that bizarre language of his, that he was eating for all the years when he had fasted. - Umberto Eco, The Name of the Rose*

## 1 Introduction

Consider two users making changes via their local clients to a document held on a remote server. It is the job of the software running on both the clients and the server to merge these changes somehow. The `diff3` algorithm, recently formalised [2], will attempt to merge these changes by comparing them with the original document. It is designed to ensure that any changes that are merged without further recourse to the user do not conflict. If conflicts do occur, these are flagged and reported to both users, one of whom must make the necessary efforts to resolve them.

What if such an opportunity for user intervention is not possible or appropriate, however? It is possible to merge all changes without conflict under the most general conditions? We show that the answer is a qualified yes, with two provisos. Firstly, the changes need to be handled in order. Secondly, it must be accepted that the result of merging the changes may not make immediate sense. This is the inevitable consequence of avoiding all conflicts, but there are situations when this is acceptable. Google Docs [1], for example, provides a collaborative tool for editing documents in real time. A document's integrity, enshrined by the `diff3` algorithm, is passed over in favour of a more direct approach. Here we formalise a similar approach and prove that, given the above provisos, any number of concurrent changes to a document by any number of users can be merged without conflicts and at a relatively low computational cost.

## 2 Background

Suppose users 1 and 2 begin with the same instance  $s$  of a document, and suppose both insert the character 'a' at differing positions, user 1 at position 12 and user 2 at position 27. When each character is inserted, the client generates a "diff" representing the insert, namely  $i(12, a)$  and  $i(27, a)$  for users 1 and 2, respectively. Each user now sees a different document, which we represent by way of the original instance of the document with the relevant diff applied to it, so  $s * i(12, a)$  and  $s * i(27, a)$  for users 1 and 2, respectively.

It is clear that each client cannot necessarily apply the other user's diff without first amending it. The  $i(27, a)$  diff cuts the original document  $s$  at position 27, for example, and if it is to affect the document  $s * i(12, a)$  in the same way it must cut it at position 28. We say that the  $i(12, a)$  diff "lifts" the  $i(27, a)$  diff and write  $i(28, a) = i(27, a) \uparrow i(12, a)$ . On the other hand, the  $i(12, a)$

diff cuts the original document  $s$  at position 12 and if it is to affect the document  $s * i(27, a)$  in the same way it must still cut it at position 12. There is no need for it to be lifted, therefore. These amended diffs, one lifted, the other left unchanged, can then be applied to the respective documents with the end result being the same:

$$s * i(12, a) * i(28, a) = s * i(27, a) * i(12, a) \quad (2.1)$$

We can extend the concept of lifting to deletes. Suppose that user 1 deletes the 5th character from the original document  $s$  and user 2 deletes the 16th character. The clients again generate diffs, this time  $d(5, 1)$  and  $d(16, 1)$  for users 1 and 2, respectively. If we think along the same lines as the previous argument we see that the  $d(16, 1)$  diff will leave the  $d(5, 1)$  diff unchanged whereas the  $d(5, 1)$  diff will lift the  $d(16, 1)$  diff. This diff cuts the original document  $s$  at position 16 but must cut the document  $s * d(5, 1)$  at position 15 if it is to affect it in the same way. This time we write  $d(15, 1) = d(16, 1) \uparrow d(5, 1)$ . As in the previous argument these two amended diffs, one lifted, the other left unchanged, can then be applied to the respective documents with the end result being the same:

$$s * d(5, 1) * d(15, 1) = s * d(16, 1) * d(5, 1) \quad (2.2)$$

Identities 2.1 and 2.2 are clearly instances of a more general rule and in what follows we show that this rule holds under for the moment what are quite restrictive conditions. We begin by assuming that the two diffs in question do not clash. In the case of deletes it is clear what this means. If the parts of the document that are deleted overlap, the deletes are said to clash. In the case of inserts, since the content of the document remains intact but is merely shifted, it could be argued the clashes never occur. For the moment we adopt the same rules as that for deletes, however. If two inserts overlap, they clash. Combinations of inserts and deletes are treated similarly. We can now define lifting for diffs that do not clash.

**Definition 2.1.**

$$\begin{aligned} i(n_2, s_2) \uparrow i(n_1, s_1) &= i(n_2 + |s_1|, s_2) & n_1 + |s_1| &\leq n_2 \\ i(n_2, s_2) \uparrow d(n_1, l_1) &= i(n_2 - l_1, s_2) & n_1 + l_1 &\leq n_2 \\ d(n_2, l_2) \uparrow i(n_1, s_1) &= d(n_2 + |s_1|, l_2) & n_1 + |s_1| &\leq n_2 \\ d(n_2, l_2) \uparrow d(n_1, l_1) &= d(n_2 - l_1, l_2) & n_1 + l_1 &\leq n_2 \end{aligned}$$

Note that lifting is only defined when the diff to be lifted is strictly to the right of the diff that does the lifting. To continue, in order to formulate a general rule, we must define the concept of one diff affecting the other, even if the effect is to leave the other unchanged.

**Definition 2.2.**

$$\begin{aligned} i(n_1, s_1)(i(n_2, s_2)) &= \begin{cases} i(n_2, s_2) & n_1 \geq n_2 + |s_2| \\ i(n_2, s_2) \uparrow i(n_1, s_1) & n_1 + |s_1| \geq n_2 \end{cases} \\ d(n_1, l_1)(i(n_2, s_2)) &= \begin{cases} i(n_2, s_2) & n_1 \geq n_2 + |s_2| \\ i(n_2, s_2) \uparrow d(n_1, l_1) & n_1 + l_1 \geq n_2 \end{cases} \\ i(n_1, s_1)(d(n_2, l_2)) &= \begin{cases} d(n_2, l_2) & n_1 \geq n_2 + l_2 \\ d(n_2, l_2) \uparrow i(n_1, s_1) & n_1 + |s_1| \geq n_2 \end{cases} \\ d(n_1, l_1)(d(n_2, l_2)) &= \begin{cases} d(n_2, l_2) & n_1 \geq n_2 + l_2 \\ d(n_2, l_2) \uparrow d(n_1, l_1) & n_1 + l_1 \geq n_2 \end{cases} \end{aligned}$$

We define the substring  $s[n...m]$  to be the string formed by taking the  $n$ 'th to the  $m$ 'th characters of the string  $s$  inclusive. We also make use of the abbreviations  $s[...m] = s[0...m]$  and  $s[n...] = s[n...|s|]$  where  $|s|$  is the length of the string. We can now formulate a general rule.

**Lemma 2.1.** *Assuming  $(n_1 \geq n_2 + |s_2|) \wedge (n_1 + |s_1| \leq n_2)$ ,  $(n_1 \geq n_2 + |s_2|) \wedge (n_1 + l_1 \leq n_2)$ , etc, we have, respectively:*

$$\begin{aligned}
s * i(n_1, s_1) * i(n_1, s_1)(i(n_2, s_2)) &= s * i(n_2, s_2) * i(n_2, s_2)(i(n_1, s_1)) \\
s * d(n_1, l_1) * d(n_1, l_1)(i(n_2, s_2)) &= s * i(n_2, s_2) * i(n_2, s_2)(d(n_1, l_1)) \\
s * i(n_1, s_1) * i(n_1, s_1)(d(n_2, l_2)) &= s * d(n_2, l_2) * d(n_2, l_2)(i(n_1, s_1)) \\
s * d(n_1, l_1) * d(n_1, l_1)(d(n_2, l_2)) &= s * d(n_2, l_2) * d(n_2, l_2)(d(n_1, l_1))
\end{aligned}$$

*Proof.* We prove the first identity when  $n_1 \geq n_2 + |s_2|$ , which gives  $i(n_1, s_1)(i(n_2, s_2)) = i(n_2, s_2)$  and  $i(n_2, s_2)(i(n_1, s_1)) = i(n_1 + |s_2|, s_2)$ . Then:

$$\begin{aligned}
s * i(n_1, s_1) * i(n_1, s_1)(i(n_2, s_2)) &= s * i(n_1, s_1) * i(n_2, s_2) \\
&= (s[\dots n_1 - 1] + s_1 + s[n_1 \dots]) * i(n_2, s_2) \\
&= s[\dots n_2 = 1] + s_2 + s[n_2 \dots n_1 - 1] + s_1 + s[n_1 \dots] \\
&= (s[\dots n_2 - 1] + s_2 + s[n_2 \dots]) * i(n_1 + |s_2|, s_1) \\
&= s * i(n_2, s_2) * i(n_2, s_2)(i(n_1, s_1))
\end{aligned}$$

The other seven cases are entirely similar.  $\square$

Note that this rule holds only when the two diffs are entirely separate, with one diff's affect on the other not being defined when  $n_1 < n_2 + |s_2|$  and  $n_2 < n_1 + |s_1|$ . We address this deficiency in the next section.

We now consider the two scenarios resulting from the differing orders in which the diffs are put on the server, shown in figures 1 and 2, with the outer strands representing the client states and the two inner strands representing the server state.

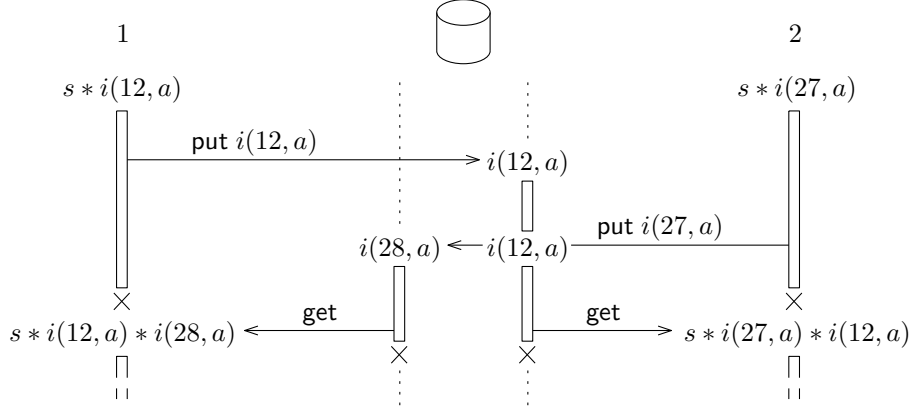


Figure 1: A put results in the put insert being amended

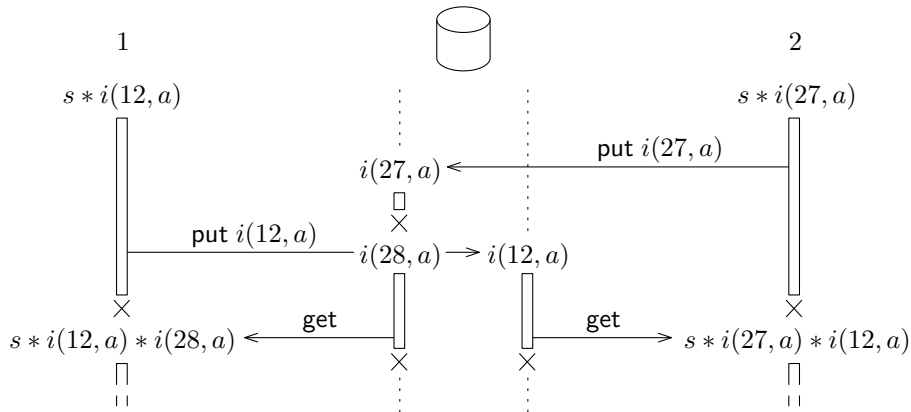


Figure 2: A put results in an existing insert being amended

The first scenario is shown in figure 1. Here the  $i(12, a)$  diff is put on the server first. When the  $i(27, a)$  diff is put on the server, it must be lifted. The second scenario is shown in figure 2. Here the  $i(27, a)$  diff is put on the server first. When the  $i(12, a)$  diff is put on the server, it is still the  $i(27, a)$  diff that must be lifted. To summarise, it is the  $i(27, a)$  diff that is lifted regardless of the order in which the diffs are put on the server. Assuming that the diffs do not clash, it is clear that when a diff is put on the server it must be compared to the existing diff on the other stack, if there is one, and that one or other must be lifted as a result.

We have defined the effect of one diff on another as being either lifting or leaving it unchanged and we therefore conclude that when a diff is put on the server and a diff already exists there, both are affected, each by the other. Although this conclusion seems somewhat strained because at this point we have only considered the most simple cases, it turns out to be fundamental. We formalise it in a more abstract way before moving on.

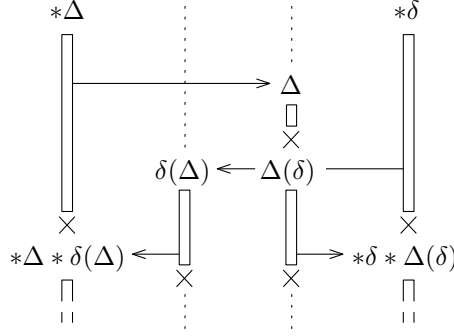


Figure 3: An abstract representation of diffs being put on the server

Let  $\Delta$  and  $\delta$  be arbitrary diffs. When we make the distinction at all, we say that  $\Delta$  is put on the server first. As a result of  $\delta$  being put on the server we have concluded that both  $\Delta$  and  $\delta$  must be amended, with  $\Delta$  becoming  $\delta(\Delta)$  and  $\delta$  becoming  $\Delta(\delta)$ . Figure 3 shows the construction. We are not particularly interested in the document that these diffs affect and so we omit this. As a result of the amendments to the diffs we expect the following identity to hold:

$$* \Delta * \Delta(\delta) = * \delta * \delta(\Delta) \quad (2.3)$$

We have proved that this identity holds in the most simple cases. In the next section we prove it formally for all cases and in the sections that follow, extend it to many diffs and many users.

### 3 A formal treatment for single diffs

Let  $\Sigma$  be a non-empty, finite set of characters from some alphabet, ranged over by  $\sigma$ . A string is any finite or countable sequence of characters from  $\Sigma$ , ranged over by  $s, s'$  and so on. The length of a string  $s$ , written  $|s|$ , is the length of this sequence. The set of these strings is written  $\Sigma^*$  and the set of non-empty strings  $\Sigma^+$ . Substrings are as previously defined. The last character of a string  $s$  is written  $s[-1]$  if the string is not of zero length. We write  $\sigma^n$  for a string of identical characters and  $\sigma^\omega$  for such a string of infinite length. We write  $s' + s''$  for the concatenation of strings  $s'$  and  $s''$ . We usually omit the  $+$  sign for strings of the form  $\sigma^n$ .

**Definition 3.1.** *The diffs  $\Delta, \delta$  and so on range over the following set:*

$$\{i(n, s) | n \in \mathbb{N}, s \in \Sigma^+\} \cup \{d(n, l) | n \in \mathbb{N}, l \in \mathbb{N}^+\}$$

Intuitively  $i(n, s)$  is an insert,  $d(n, l)$  a delete. We say that the insert  $i(n, s)$  employs the string  $s$ .

**Definition 3.2.** *The diff  $e$  ranges over the following set:*

$$\{e()\}$$

Intuitively  $e()$  is the empty diff.

**Definition 3.3.**

$$\begin{aligned}\lrcorner i(n, s) &= n \ i(n, s) \lrcorner = n + |s| - 1 \\ \lrcorner d(n, l) &= n \ d(n, l) \lrcorner = n + l - 1\end{aligned}$$

**Definition 3.4.**

$$\begin{aligned}\Delta \simeq \delta &\text{ iff } \lrcorner \Delta = \lrcorner \delta \quad \Delta ! \delta \text{ iff } (\lrcorner \Delta \leq \lrcorner \delta) \wedge (\Delta \geq \lrcorner \delta) \\ \Delta \ll \delta &\text{ iff } \Delta \lrcorner < \lrcorner \delta \quad \Delta < \delta \text{ iff } (\lrcorner \Delta < \lrcorner \delta) \wedge (\Delta \geq \lrcorner \delta) \\ \Delta \gg \delta &\text{ iff } \lrcorner \Delta > \lrcorner \delta \quad \Delta > \delta \text{ iff } (\lrcorner \Delta \leq \lrcorner \delta) \wedge (\Delta > \lrcorner \delta)\end{aligned}$$

**Lemma 3.1.** *Either  $\Delta \ll \delta, \Delta < \delta, \Delta \simeq \delta, \Delta > \delta$  or  $\Delta \gg \delta$ .* □

**Lemma 3.2.**  *$\Delta ! \delta$  if and only if  $\Delta < \delta, \Delta \simeq \delta$  or  $\Delta > \delta$ .* □

We now extend lifting to include cases in which the diffs overlap.

**Definition 3.5.** *For  $n_1 \leq n_2$ :*

$$\begin{aligned}i(n_2, s_2) \uparrow i(n_1, s_1) &= i(n_2 + |s_1|, s_2) \quad d(n_2, l_2) \uparrow i(n_1, s_1) = d(n_2 + |s_1|, l_2) \\ i(n_2, s_2) \uparrow d(n_1, l_1) &= i(n_2 - l_1, s_2) \quad d(n_2, l_2) \uparrow d(n_1, l_1) = d(n_2 - l_1, l_2)\end{aligned}$$

Recall that we want to assert identity 2.3 in all cases. To begin with we treat the cases in which  $\Delta$  and  $\delta$  are both inserts.

**Definition 3.6.** *When  $\Delta$  and  $\delta$  are both inserts with  $\Delta = \delta$ ,  $\Delta(\delta) = \delta$  and  $\delta(\Delta) = \Delta$ .*

**Lemma 3.3.** *When  $\Delta$  and  $\delta$  are both inserts with  $\Delta = \delta$ ,  $*\Delta * \Delta(\delta) = *\delta * \delta(\Delta)$ .* □

Suppose that  $\Delta$  and  $\delta$  employ the strings  $s_1$  and  $s_2$ , respectively, and that these strings are of equal length but not identical. We assume that there is some lexicographical ordering on  $\Sigma^*$  so that we can say either  $s_1 < s_2$  or  $s_1 > s_2$ .

**Definition 3.7.** *When  $\Delta$  and  $\delta$  are both inserts with  $\Delta \simeq \delta$ ,  $\Delta \lrcorner = \delta \lrcorner$  but  $\Delta \neq \delta$ :*

$$\Delta(\delta) = \begin{cases} \delta \uparrow \Delta & s_2 < s_1 \\ \delta & \text{otherwise} \end{cases} \quad \delta(\Delta) = \begin{cases} \Delta \uparrow \delta & s_1 < s_2 \\ \Delta & \text{otherwise} \end{cases}$$

Intuitively we lift the lesser of the two diffs, lexicographically speaking, regardless of whether it is  $\Delta$  or  $\delta$ .

**Lemma 3.4.** *When  $\Delta$  and  $\delta$  are both inserts with  $\Delta \simeq \delta$ ,  $\Delta \lrcorner = \delta \lrcorner$  but  $\Delta \neq \delta$ ,  $*\Delta * \Delta(\delta) = *\delta * \delta(\Delta)$ .* □

In what follows, rather than consider the action of diffs on strings or resorting to the plethora of symbols just defined, we use visual proofs. We consider the action of diffs on a string of eight characters only. As an example we consider the case of two separate inserts, a case already covered in lemma 2.1. The figure below shows the construction, with both inserts lined up behind the string, ready to be applied to it. On the left, as  $\Delta$  is applied first and since  $\delta$  is strictly to the right of  $\Delta$ , it must be lifted. In the final step, the lifted diff is then applied. On the right the diffs are reversed, with the end result being the same. Note that this time  $\Delta$  is not lifted, since it is strictly to the left of  $\delta$ .

$\begin{array}{c}   + +   \\   + + +   \\ a \ b \ c \ d \ e \ f \ g \ h \end{array}$	$\begin{array}{c} \delta \\ \Delta \end{array}$	$\begin{array}{c}   + + +   \\   + +   \\ a \ b \ c \ d \ e \ f \ g \ h \end{array}$	$\begin{array}{c} \Delta \\ \delta \end{array}$
$\begin{array}{c}   + +   \\   \cdot \cdot \cdot   a \ b \ c \ d \ e \ f \ g \ h \\   \cdot \cdot \cdot   a \ b \ c \ d   \cdot \cdot \cdot   e \ f \ g \ h \end{array}$	$\begin{array}{c} \delta \uparrow \Delta \end{array}$	$\begin{array}{c}   + + +   \\ a \ b \ c \ d   \cdot \cdot \cdot   e \ f \ g \ h \\   \cdot \cdot \cdot   a \ b \ c \ d   \cdot \cdot \cdot   e \ f \ g \ h \end{array}$	$\begin{array}{c} \Delta \end{array}$

Two inserts	$\Delta \ll \delta, \Delta < \delta$		
+ +	$\delta$	+ + +	$\Delta$
+ + +	$\Delta$	+ +	$\delta$
a b c d e f g h		a b c d e f g h	
+ +	$\delta \uparrow \Delta$	+ + +	$\Delta$
. . .  a b c d e f g h		a b . .  c d e f g h	
. . .  a b . .  c d e f g h		. . .  a b . .  c d e f g h	
	$\therefore * \Delta * (\delta \uparrow \Delta) = * \delta * \Delta$		

Two inserts	$\Delta > \delta, \Delta \gg \delta$		
+ +	$\delta$	+ + +	$\Delta$
+ + +	$\Delta$	+ +	$\delta$
a b c d e f g h		a b c d e f g h	
+ +	$\delta$	+ + +	$\Delta \uparrow \delta$
a b . .  c d e f g h		. .  a b c d e f g h	
. .  a b . .  c d e f g h		. .  a b . .  c d e f g h	
	$\therefore * \Delta * \delta = * \delta * (\Delta \uparrow \delta)$		

Figure 4: Two inserts

Employing specific strings does not clarify the proofs and instead the diffs are distinguished by their lengths. The fact that the end result is the same both left and right constitutes the proof. Although informal, these visual proofs capture the essence of each argument and as the cases become more subtle they become indispensable. We begin by proving the remaining cases for two inserts in this manner.

**Definition 3.8.** When  $\Delta$  and  $\delta$  are both inserts with  $\Delta \ll \delta$  or  $\Delta < \delta$ ,  $\Delta(\delta) = (\delta \uparrow \Delta)$  and  $\delta(\Delta) = \Delta$ .

**Definition 3.9.** When  $\Delta$  and  $\delta$  are both inserts with  $\Delta > \delta$  or  $\Delta \gg \delta$ ,  $\Delta(\delta) = \delta$  and  $\delta(\Delta) = (\Delta \uparrow \delta)$ .

**Lemma 3.5.** When  $\Delta$  and  $\delta$  are both inserts with  $\Delta \ll \delta$ ,  $\Delta < \delta$ ,  $\Delta > \delta$  or  $\Delta \gg \delta$ ,  $*\Delta * \Delta(\delta) = *\delta * \delta(\Delta)$ .

*Proof.* See figure 4. □

We now treat some of the cases when one diff is an insert and the other a delete.

**Definition 3.10.** When  $\Delta$  is an insert and  $\delta$  a delete with  $\Delta \ll \delta$ ,  $\Delta \simeq \delta$  or  $\Delta < \delta$ ,  $\Delta(\delta) = (\delta \uparrow \Delta)$  and  $\delta(\Delta) = \Delta$ .

**Definition 3.11.** When  $\Delta$  is an insert and  $\delta$  a delete with  $\Delta \gg \delta$ ,  $\Delta(\delta) = \delta$  and  $\delta(\Delta) = (\Delta \uparrow \delta)$ .

**Lemma 3.6.** When  $\Delta$  is an insert and  $\delta$  a delete with  $\Delta \ll \delta$ ,  $\Delta < \delta$ ,  $\Delta \simeq \delta$  or  $\Delta \gg \delta$ ,  $*\Delta * \Delta(\delta) = *\delta * \delta(\Delta)$ .

*Proof.* See figure 5. □

**Definition 3.12.** When  $\Delta$  is a delete and  $\delta$  an insert with  $\Delta \ll \delta$ ,  $\Delta(\delta) = (\delta \uparrow \Delta)$  and  $\delta(\Delta) = \Delta$ .

**Definition 3.13.** When  $\Delta$  is a delete and  $\delta$  an insert with  $\Delta \ll \delta$ ,  $\Delta \simeq \delta$  or  $\Delta > \delta$ ,  $\Delta(\delta) = \delta$  and  $\delta(\Delta) = (\Delta \uparrow \delta)$ .

**Lemma 3.7.** When  $\Delta$  is a delete and  $\delta$  an insert with  $\Delta \ll \delta$ ,  $\Delta \simeq \delta$ ,  $\Delta > \delta$  or  $\Delta \gg \delta$ ,  $*\Delta * \Delta(\delta) = *\delta * \delta(\Delta)$ .

*Proof.* See figure 5. □

We now treat the two easiest cases when  $\Delta$  and  $\delta$  are both deletes.

**Definition 3.14.** When  $\Delta$  and  $\delta$  are both deletes with  $\Delta = \delta$ ,  $\Delta(\delta) = \epsilon$  and  $\delta(\Delta) = \Delta$ .

**Definition 3.15.** When  $\Delta$  and  $\delta$  are both deletes with  $\Delta \ll \delta$ ,  $\Delta(\delta) = (\delta \uparrow \Delta)$  and  $\delta(\Delta) = \Delta$ .

**Definition 3.16.** When  $\Delta$  and  $\delta$  are both deletes with  $\Delta \gg \delta$ ,  $\Delta(\delta) = \delta$  and  $\delta(\Delta) = (\Delta \uparrow \delta)$ .

**Lemma 3.8.** When  $\Delta$  and  $\delta$  are both deletes with  $\Delta \ll \delta$  or  $\Delta \gg \delta$ ,  $*\Delta * \Delta(\delta) = *\delta * \delta(\Delta)$ .

*Proof.* See figure 6. □

An additional definition is needed in order to treat the majority of the remaining cases.

Insert then delete		$\Delta \ll \delta, \Delta < \delta$	
$\begin{array}{c}   \cdot -   \\   + + +   \\ a b c d e f g h \end{array}$	$\delta$	$\begin{array}{c}   + + +   \\   \cdot -   \\ a b c d e f g h \end{array}$	$\Delta$
$\begin{array}{c}   \cdot -   \\ a b   \cdot \cdot \cdot   c d e f g h \\ a b   \cdot \cdot \cdot   c d e h \end{array}$	$\delta \uparrow \Delta$	$\begin{array}{c}   + + +   \\ a b c d e h \end{array}$	$\Delta$
$\therefore * \Delta * (\delta \uparrow \Delta) = * \delta * \Delta$			

Insert then delete		$\Delta \gg \delta$	
$\begin{array}{c}   \cdot - \cdot -   \\   + +   \\ a b c d e f g h \end{array}$	$\delta$	$\begin{array}{c}   + +   \\   \cdot - \cdot -   \\ a b c d e f g h \end{array}$	$\Delta$
$\begin{array}{c}   \cdot - \cdot -   \\ a b c d   \cdot \cdot \cdot   e f g h \\ a   \cdot \cdot \cdot   e f g h \end{array}$	$\delta$	$\begin{array}{c}   + +   \\ a e f g h \end{array}$	$\Delta \uparrow \delta$
$\therefore * \Delta * \delta = * \delta * (\Delta \uparrow \delta)$			

Delete then insert		$\Delta \ll \delta$	
$\begin{array}{c}   + + +   \\   \cdot -   \\ a b c d e f g h \end{array}$	$\delta$	$\begin{array}{c}   \cdot -   \\   + + +   \\ a b c d e f g h \end{array}$	$\Delta$
$\begin{array}{c}   + + +   \\ a d e f g h \end{array}$	$\delta \uparrow \Delta$	$\begin{array}{c}   \cdot -   \\ a b c d   \cdot \cdot \cdot   e f g h \end{array}$	$\Delta$
$\therefore * \Delta * (\delta \uparrow \Delta) = * \delta * \Delta$			

Delete then insert		$\Delta \simeq \delta$	
$\begin{array}{c}   + + +   \\   \cdot -   \\ a b c d e f g h \end{array}$	$\delta$	$\begin{array}{c}   \cdot -   \\   + + +   \\ a b c d e f g h \end{array}$	$\Delta$
$\begin{array}{c}   + + +   \\ a d e f g h \end{array}$	$\delta$	$\begin{array}{c}   \cdot -   \\ a   \cdot \cdot \cdot   b c d e f g h \end{array}$	$\Delta \uparrow \delta$
$\therefore * \Delta * \delta = * \delta * (\Delta \uparrow \delta)$			

Delete then insert		$\Delta > \delta, \Delta \gg \delta$	
$\begin{array}{c}   + +   \\   \cdot - \cdot -   \\ a b c d e f g h \end{array}$	$\delta$	$\begin{array}{c}   \cdot - \cdot -   \\   + +   \\ a b c d e f g h \end{array}$	$\Delta$
$\begin{array}{c}   + +   \\ a b f g h \end{array}$	$\delta$	$\begin{array}{c}   \cdot - \cdot -   \\ a   \cdot \cdot \cdot   b c d e f g h \end{array}$	$\Delta \uparrow \delta$
$\therefore * \Delta * \delta = * \delta * (\Delta \uparrow \delta)$			

Figure 5: An insert and a delete in either order

**Definition 3.17.** When  $\Delta$  and  $\delta$  are both deletes with  $\Delta < \delta$ ,  $(\delta - \Delta)$  is such that  $\lfloor (\delta - \Delta) = \Delta_{\downarrow} + 1, (\delta - \Delta)_{\downarrow} = \delta_{\downarrow}$  and when  $\Delta > \delta$ ,  $(\Delta - \delta)$  is such that  $\lfloor (\Delta - \delta) = \delta_{\downarrow} + 1, (\Delta - \delta)_{\downarrow} = \Delta_{\downarrow}$ .

Intuitively, the portion of  $\delta$  overlapping  $\Delta$  is discarded in order to form  $(\delta - \Delta)$  and vice versa.

**Definition 3.18.** When  $\Delta$  and  $\delta$  are both deletes with  $\Delta < \delta$  and  $\Delta_{\downarrow} < \delta_{\downarrow}$ ,  $\Delta(\delta) = ((\delta - \Delta) \uparrow \Delta)$  and  $\delta(\Delta) = (\Delta - \delta)$ .

**Definition 3.19.** When  $\Delta$  and  $\delta$  are both deletes with  $\Delta \simeq \delta$  and  $\Delta_{\downarrow} < \delta_{\downarrow}$ ,  $\Delta(\delta) = ((\delta - \Delta) \uparrow \Delta)$  and  $\delta(\Delta) = \epsilon$ .

**Definition 3.20.** When  $\Delta$  and  $\delta$  are both deletes with  $\Delta \simeq \delta$  and  $\Delta_{\downarrow} > \delta_{\downarrow}$ ,  $\Delta(\delta) = \epsilon$  and  $\delta(\Delta) = ((\Delta - \delta) \uparrow \delta)$ .

**Definition 3.21.** When  $\Delta$  and  $\delta$  are both deletes with  $\Delta > \delta$  and  $\Delta_{\downarrow} > \delta_{\downarrow}$ ,  $\Delta(\delta) = (\delta - \Delta)$  and  $\delta(\Delta) = ((\Delta - \delta) \uparrow \delta)$ .

**Lemma 3.9.** When  $\Delta$  and  $\delta$  are both deletes with  $\Delta < \delta$  and  $\Delta_{\downarrow} < \delta_{\downarrow}$ ;  $\Delta \simeq \delta$  and  $\Delta_{\downarrow} < \delta_{\downarrow}$ ;  $\Delta \simeq \delta$  and  $\Delta_{\downarrow} > \delta_{\downarrow}$ ; or  $\Delta > \delta$  and  $\Delta_{\downarrow} > \delta_{\downarrow}$ ,  $* \Delta * \Delta(\delta) = * \delta * \delta(\Delta)$ .

*Proof.* See figure 7. □

In order to treat all of the remaining cases, one further definition is needed.

**Definition 3.22.** For diffs  $d(n_1, l_1)$  or  $i(n_1, s_1)$  together with  $d(n_2, l_2)$ , if  $n_2 < n_1 < n_2 + l + 2 - 1$  we define:

$$d(n_2, l_2)^- = d(n_2, n_1 - n_2) d(n_2, l_2)^+ = d(n_2, l_2 - n_1 + n_2)$$

Two deletes	$\Delta \ll \delta$		
$\begin{array}{c}   \cdot \cdot   \\   \cdot \cdot \cdot   \\ a b c d e f g h \end{array}$	$\begin{array}{c} \delta \\ \Delta \end{array}$	$\begin{array}{c}   \cdot \cdot \cdot   \\   \cdot \cdot \cdot   \\ a b c d e f g h \end{array}$	$\begin{array}{c} \Delta \\ \delta \end{array}$
$\begin{array}{c}   \cdot \cdot   \\ d e f g h \end{array}$	$\delta \uparrow \Delta$	$\begin{array}{c}   \cdot \cdot \cdot   \\ a b c d g h \end{array}$	$\Delta$
$d g h$		$d g h$	
$\therefore * \Delta * (\delta \uparrow \Delta) = * \delta * \Delta$			

Two deletes	$\Delta \gg \delta$		
$\begin{array}{c}   \cdot \cdot   \\   \cdot \cdot \cdot   \\ a b c d e f g h \end{array}$	$\begin{array}{c} \delta \\ \Delta \end{array}$	$\begin{array}{c}   \cdot \cdot \cdot   \\   \cdot \cdot \cdot   \\ a b c d e f g h \end{array}$	$\begin{array}{c} \Delta \\ \delta \end{array}$
$\begin{array}{c}   \cdot \cdot   \\ a b c d h \end{array}$	$\delta$	$\begin{array}{c}   \cdot \cdot \cdot   \\ a d e f g h \end{array}$	$\Delta \uparrow \delta$
$a d h$		$a d h$	
$\therefore * \Delta * \delta = * \delta * (\Delta \uparrow \delta)$			

Figure 6: Two easy cases for two deletes

Intuitively, a delete is split by another delete or insert. In fact if the diff  $\delta$  is split by  $\Delta$  then  $\delta^-$  is no more than  $(\delta - \Delta)$  with the  $\delta^+$  part being the part of  $\delta$  that is discarded. Note that when defining the “double diff”  $(\delta^-; \delta^+)$ , the diff that splits  $\delta$  is left implicit, but it is always clear from the context. Once formed, the single diffs  $\delta^+$  and  $\delta^-$  that comprise the double diff are often amended and therefore it is useful to consider the action of a double diff  $(\delta', \delta'')$  for more or less arbitrary diffs  $\delta'$  and  $\delta''$ .

**Definition 3.23.** For diffs  $\delta'$  and  $\delta''$ , as long as  $(\delta'' \uparrow \delta')$  is defined, we define:

$$*(\delta'; \delta'') = * \delta' * (\delta'' \uparrow \delta')$$

**Lemma 3.10.** If  $(\delta^-; \delta^+)$  is formed from  $\delta$  then  $*(\delta^-; \delta^+) = * \delta$ . □

We are now in a position to prove the remaining cases.

**Definition 3.24.** When  $\Delta$  is an insert and  $\delta$  a delete with  $\Delta > \delta$ ,  $\Delta(\delta) = (\delta^-; \delta^+ \uparrow \Delta)$  and  $\delta(\Delta) = (\Delta \uparrow \delta^-)$ .

**Definition 3.25.** When  $\Delta$  is a delete and  $\delta$  an insert with  $\Delta < \delta$ ,  $\Delta(\delta) = (\delta \uparrow \Delta^-)$  and  $\delta(\Delta) = (\Delta^-; \Delta^+ \uparrow \Delta)$ .

**Definition 3.26.** When  $\Delta$  and  $\delta$  are both deletes with  $\Delta > \delta$  and  $\Delta_{\downarrow} < \delta_{\downarrow}$ ,  $\Delta(\delta) = (\delta^-; (\delta^+ - \Delta) \uparrow \Delta)$  and  $\delta(\Delta) = \epsilon$ .

**Definition 3.27.** When  $\Delta$  and  $\delta$  are both deletes with  $\Delta < \delta$  and  $\Delta_{\downarrow} > \delta_{\downarrow}$ ,  $\Delta(\delta) = \epsilon$  and  $\delta(\Delta) = (\Delta^-; (\Delta^+ - \delta) \uparrow \delta)$ .

**Lemma 3.11.** If  $\Delta$  is an insert and  $\delta$  a delete with  $\Delta > \delta$ ;  $\Delta$  is a delete and  $\delta$  a insert with  $\Delta < \delta$ ;  $\Delta$  and  $\delta$  are both deletes with  $\Delta < \delta$  and  $\Delta_{\downarrow} > \delta_{\downarrow}$ ; or  $\Delta$  and  $\delta$  are both deletes with  $\Delta > \delta$  and  $\Delta_{\downarrow} < \delta_{\downarrow}$ , then  $* \Delta * \Delta(\delta) = * \delta * \delta(\Delta)$ .

*Proof.* See figure 8. □

All the cases have now been covered.

**Theorem 3.1.** If  $\Delta$  and  $\delta$  are any two diffs, then  $* \Delta * \Delta(\delta) = * \delta * \delta(\Delta)$ . □

## 4 A attempted treatment for many diffs

To begin with we redefine  $\Delta$  and  $\delta$  to be arrays of diffs, setting  $\Delta = [\Delta_0, \Delta_1 \dots]$  and  $\delta = [\delta_0, \delta_1 \dots]$  with  $\Delta[0] = \Delta_0$ ,  $\Delta[1] = \Delta_1$ ,  $\delta[0] = \delta_0$ ,  $\delta[1] = \delta_1$  and so on. Defining the action of an array of diffs is straightforward and we note that the concept of lifting plays no part in this.

**Definition 4.1.**

$$s * [\Delta_0, \Delta_1 \dots] = (s * \Delta_0) * \Delta_1 \dots$$

Application is from left to right, for example  $(s * \Delta_0) * \Delta_1$  is written  $s * \Delta_0 * \Delta_1$ , and we drop the parentheses whenever possible from now on. It is helpful to couch the above definition in recursive form.

Two deletes $\Delta < \delta, \Delta_{\downarrow} \leq \delta_{\downarrow}$			
$ - - - - $	$\delta$	$ - - - $	$\Delta$
$ - - - $	$\Delta$	$ - - - - $	$\delta$
a b c d e f g h		a b c d e f g h	
$ - - - - $	$\delta - \Delta$	$ - - - $	$\Delta - \delta$
$ - - - $	$\Delta$	$ - - - $	$\delta$
a b c d e f g h		a b c d e f g h	
$ - - - $	$(\delta - \Delta) \uparrow \Delta$	$ - - - $	$\Delta - \delta$
d e f g h		d e f g h	
g h		g h	
$\therefore * \Delta * ((\delta - \Delta) \uparrow \Delta) = * \delta * (\Delta - \delta)$			

Two deletes $\Delta \simeq \delta, \Delta_{\downarrow} < \delta_{\downarrow}$			
$ - - - - $	$\delta$	$ - - - $	$\Delta$
$ - - - $	$\Delta$	$ - - - - $	$\delta$
a b c d e f g h		a b c d e f g h	
$ - - $	$\delta - \Delta$		$\epsilon$
$ - - - $	$\Delta$	$ - - - - $	$\delta$
a b c d e f g h		a b c d e f g h	
$ - $	$(\delta - \Delta) \uparrow \Delta$		$\epsilon$
d e f g h		e f g h	
e f g h		e f g h	
$\therefore * \Delta * ((\delta - \Delta) \uparrow \Delta) = * \delta * \epsilon$			

Two deletes $\Delta \simeq \delta, \Delta_{\downarrow} > \delta_{\downarrow}$			
$ - - - $	$\delta$	$ - - - - $	$\Delta$
$ - - - - $	$\Delta$	$ - - - $	$\delta$
a b c d e f g h		a b c d e f g h	
	$\epsilon$	$ - $	$\Delta - \delta$
$ - - - - $	$\Delta$	$ - - - $	$\delta$
a b c d e f g h		a b c d e f g h	
	$\epsilon$	$ - $	$(\Delta - \delta) \uparrow \delta$
e f g h		d e f g h	
e f g h		e f g h	
$\therefore * \Delta * \epsilon = * \delta * ((\Delta - \delta) \uparrow \delta)$			

Two deletes $\Delta > \delta, \Delta_{\downarrow} \geq \delta_{\downarrow}$			
$ - - - $	$\delta$	$ - - - - $	$\Delta$
$ - - - - $	$\Delta$	$ - - - $	$\delta$
a b c d e f g h		a b c d e f g h	
$ - $	$\delta - \Delta$	$ - - $	$\Delta - \delta$
$ - - - - $	$\Delta$	$ - - - $	$\delta$
a b c d e f g h		a b c d e f g h	
$ - $	$\delta - \Delta$	$ - - $	$(\Delta - \delta) \uparrow \delta$
a b c h		a b f g h	
a b h		a b h	
$\therefore * \Delta * (\delta - \Delta) = * \delta * ((\Delta - \delta) \uparrow \delta)$			

Figure 7: Four harder cases for two deletes

**Definition 4.2.**

$$s * \Delta = \begin{cases} s * \Delta[0] & |\Delta| = 1 \\ s * \Delta[0] * \Delta[1...] & |\Delta| > 1 \end{cases}$$

We define lifting for arrays of diffs but must proceed with care. In particular, the following definitions and lemmas are only valid when single diffs are involved.

**Definition 4.3.** *Provided that  $\delta_0, \Delta_0(\delta_0), \Delta_1(\Delta_0(\delta_0))$  and so on are all single diffs, we define, for  $|\delta| = 1$ :*

$$[\Delta_0, \Delta_1...] \circ \delta = [...\Delta_1(\Delta_0(\delta_0))]$$

Note that the result is an array of diffs of length 1. Again it is helpful to couch this definition recursively.

**Definition 4.4.** *Provided that  $\delta[0]$  and  $\Delta[0](\delta[0])$  are single diffs, we define for  $|\delta| = 1$*

$$\Delta \circ \delta = \begin{cases} [\Delta[0](\delta[0])] & |\Delta| = 1 \\ \Delta[1...] \circ [\Delta[0](\delta[0])] & |\Delta| > 1 \end{cases}$$

**Definition 4.5.** *If  $\Delta \circ \delta$  is defined, we define  $\delta \circ \Delta$  for  $|\delta| = 1$  and  $|\Delta| \geq 1$  to be the array of diffs such that the following identity holds:*

$$* \Delta * \Delta \circ \delta = * \delta * \delta \circ \Delta$$

We can derive an explicit definition but again must proceed with care and avoid all double diffs.

**Lemma 4.1.** *Provided that  $\delta(\Delta[0])$  and  $\Delta[0](\delta)$  are single diffs, the following definition holds for  $|\delta| = 1$ :*

$$\delta \circ \Delta = \begin{cases} [\delta[0](\Delta[0])] & |\Delta| = 1 \\ [\delta[0](\Delta[0])] + [\Delta[0](\delta[0])] \circ \Delta[1...] & |\Delta| > 1 \end{cases}$$

Insert then delete		$\Delta > \delta$	
- - - -	$\delta$	+ + +	$\Delta$
+ + +	$\Delta$	- - - -	$\delta$
a b c d e f g h		a b c d e f g h	
- - - -	$\delta^+$	+ + +	$\Delta$
-	$\delta^-$	- - - -	$\delta^+$
+ + +	$\Delta$	-	$\delta^-$
a b c d e f g h		a b c d e f g h	
- - - -	$\delta^+ \uparrow \Delta$	+ + +	$\Delta \uparrow \delta^-$
-	$\delta^-$	- - - -	$\delta^+ \uparrow \delta^-$
a b   . . .   c d e f g h		a c d e f g h	
- - - -	$(\delta^+ \uparrow \Delta) \uparrow \delta^-$	+ + +	$\Delta \uparrow \delta^-$
a   . . .   c d e f g h		a g h	
a   . . .   g h		a   . . .   g h	
$\therefore * \Delta * (\delta^-; \delta^+ \uparrow \Delta) = * (\delta^-; \delta^+) * (\Delta \uparrow \delta^-)$			
Delete then insert		$\Delta < \delta$	
+ + +	$\delta$	- - -	$\Delta$
- - -	$\Delta$	+ + +	$\delta$
a b c d e f g h		a b c d e f g h	
+ + +	$\delta$	-	$\Delta^+$
-	$\Delta^+$	- -	$\Delta^-$
- -	$\Delta^-$	+ + +	$\delta$
a b c d e f g h		a b c d e f g h	
+ + +	$\delta \uparrow \Delta^-$	-	$\Delta^+ \uparrow \delta$
-	$\Delta^+ \uparrow \Delta^-$	- -	$\Delta^-$
a d e f g h		a b c   . . .   d e f g h	
+ + +	$\delta \uparrow \Delta^-$	-	$(\Delta^+ \uparrow \delta) \uparrow \Delta^-$
a e f g h		a   . . .   d e f g h	
a   . . .   e f g h		a   . . .   e f g h	
$\therefore * (\Delta^-; \Delta^+) * (\delta \uparrow \Delta^-) = * \delta * (\Delta^-; \Delta^+ \uparrow \delta)$			
Two deletes		$\Delta > \delta, \Delta_j < \delta_j$	
- - - - -	$\delta$	- -	$\Delta$
- -	$\Delta$	- - - - -	$\delta$
a b c d e f g h		a b c d e f g h	
- - - - -	$\delta^+$	- - - - -	$\epsilon$
-	$\delta^-$	- - - - -	$\delta^+$
- -	$\Delta$	-	$\delta^-$
a b c d e f g h		a b c d e f g h	
- - -	$\delta^+ - \Delta$	- - - -	$\epsilon$
-	$\delta^-$	a c d e f g h	$\delta^+ \uparrow \delta^-$
a b c d e f g h			$\epsilon$
- - -	$(\delta^+ - \Delta) \uparrow \Delta$	a h	
-	$\delta^-$	a h	
a b e f g h			
- - -	$(\delta^+ - \Delta) \uparrow \Delta \uparrow \delta^-$		
a e f g h			
a h			
$* \Delta * (\delta^-; (\delta^+ - \Delta) \uparrow \Delta) = * (\delta^-; \delta^+) * \epsilon$			
Two deletes		$\Delta < \delta, \Delta_j > \delta_j$	
- -	$\delta$	- - - - -	$\Delta$
- - - - -	$\Delta$	- -	$\delta$
a b c d e f g h		a b c d e f l g	
- - - - -	$\epsilon$	- - - - -	$\Delta^+$
-	$\Delta^+$	-	$\Delta^-$
- -	$\Delta^-$	- -	$\delta$
a b c d e f g h		a b c d e f l g	
- - - -	$\epsilon$	- - -	$\Delta^+ - \delta$
a c d e f g h	$\Delta^+ \uparrow \Delta^-$	-	$\Delta^-$
		- -	$\delta$
a h	$\epsilon$	a b c d e f l g	
a h		- - -	$(\Delta^+ - \delta) \uparrow \delta$
		-	$\Delta^-$
		a b e f g h	
		- - -	$(\Delta^+ - \delta) \uparrow \delta \uparrow \Delta^-$
		a e f g h	
		a h	
$* \delta * (\Delta^-; (\Delta^+ - \delta) \uparrow \delta) = * (\Delta^-; \delta^+) * \epsilon$			

Figure 8: The cases involving double diffs

*Proof.* By induction on the length of  $\Delta$ . Suppose  $|\Delta| = 1$ , then:

$$* \Delta * \Delta \circ \delta = * \Delta[0] * \Delta[0](\delta[0]) = * \delta[0] * \delta[0](\Delta[0]) = * \delta[0] * [\delta[0](\Delta[0])] = * \delta * \delta \circ \Delta$$

Suppose  $|\Delta| > 1$ , with induction hypothesis  $* \Delta[1...] * \Delta[1...](\Delta[0](\delta[0])) = * \Delta[0](\delta[0]) * (\Delta[0](\delta[0])) \circ \Delta[1...]$ . Then:

$$\begin{aligned}
* \Delta * \Delta \circ \delta &= * ([\Delta[0]] + \Delta[1...]) * \Delta \circ [\delta[0]] \\
&= * ([\Delta[0]] + \Delta[1...]) * \Delta[1...] \circ [\Delta[0](\delta[0])] \\
&= * \Delta[0] * \Delta[1...] * \Delta[1...] \circ [\Delta[0](\delta[0])] \\
&= * \Delta[0] * [\Delta[0](\delta[0])] * [\Delta[0](\delta[0])] \circ \Delta[1...] \\
&= * \Delta[0] * \Delta[0](\delta[0]) * [\Delta[0](\delta[0])] \circ \Delta[1...] \\
&= * \delta[0] * \delta[0](\Delta[0]) * [\Delta[0](\delta[0])] \circ \Delta[1...] \\
&= * \delta[0] * ([\delta[0](\Delta[0])] + [\Delta[0](\delta[0])] \circ \Delta[1...]) \\
&= * \delta * \delta \circ \Delta
\end{aligned}$$

□

We now attempt to define  $\Delta \circ \delta$  in the case when  $\Delta[0]$  splits  $\delta[0]$ . To begin with we note that in the case when both diffs are deletes and a double diff is necessary, it is in fact equivalent to a single diff. We return to  $\Delta$  and  $\delta$  being single diffs in what follows.

**Lemma 4.2.** *If  $\Delta$  and  $\delta$  are both deletes with  $\Delta > \delta$  and  $\Delta_{\downarrow} < \delta_{\downarrow}$  then there exists a  $\delta'$  such that  $*\delta' = *(\delta^-; (\delta^+ - \Delta) \uparrow \Delta)$  and vice versa.*

*Proof.* See figure 8. □

The remaining cases do result in genuine double diffs but lifting them can be simplified.

**Lemma 4.3.** *In all cases  $\Delta(\delta^-; \delta^+) = (\delta^-; \Delta(\delta^+))$  and vice versa.*

*Proof.* See figure 8. □

We now attempt a definition. Suppose  $|\Delta| = 1$ :

$$\Delta \circ \delta = [\Delta[0](\delta[0])] = [\Delta[0](\delta[0]^-; \delta[0]^+)] = [(\delta[0]^-; \Delta[0](\delta[0]^+))]$$

Now suppose  $|\Delta| > 1$ :

$$\begin{aligned} \Delta \circ \delta &= \Delta[1\dots] \circ [\Delta[0](\delta[0])] \\ &= \Delta[1\dots] \circ [\Delta[0](\delta[0]^-; \delta[0]^+)] \\ &= \Delta[1\dots] \circ [(\delta[0]^-; \Delta[0](\delta[0]^+))] \end{aligned}$$

Here we must stop, having no definition for a term of the form  $\Delta \circ (\delta'; \delta'')$  where  $|\Delta| > 1$  and  $\delta', \delta''$  are arbitrary diffs. Lemma 4.2 tells us that  $(\delta^-; \Delta[0](\delta^+))$  may be replaced by a single diff in the case when  $\Delta[0]$  and  $\delta[0]$  are deletes but this is not the case in general. An alternative is to treat the lifted double diff as an array of diffs and seek a definition for terms of the form  $\Delta \circ \delta$  where  $|\Delta|, |\delta| > 1$ . This is possible but again only by carefully avoiding all double diffs, a situation that was hardly satisfactory in the first place.

## 5 A formal treatment for many diffs

Instead we take a more abstract approach. We still consider arrays of diffs  $\Delta$  and  $\delta$  but forget their specific action. We take the visual approach adopted earlier and refine it, calling the resultant diagrams ladders. In the figure below, the ladder on the left represents the situation that we are presented with initially, namely that both  $\Delta$  and  $\delta$  are applied to the same string  $s$ . On the other hand, the ideal situation is that when  $\delta$  is amended it can be applied not to  $s$  but to  $s * \Delta$ . We denote this amended diff by  $\Delta.\delta$  and represent this situation by the ladder on the right. We make it a rule that these two ladders, both representing true situations, equate to one another.

$$\frac{\frac{\delta}{\quad} s}{\frac{\Delta}{\quad} s} = \frac{\frac{\Delta.\delta}{\quad} s * \Delta}{\frac{\Delta}{\quad} s}$$

Since we are dealing with arrays of diffs we expect a rule relating to how they work and this is easily formulated:

$$\frac{\frac{\Delta' + \Delta''}{\quad} s}{\quad} = \frac{\frac{\Delta''}{\quad} s * \Delta'}{\frac{\Delta'}{\quad} s}$$

For the sake of completeness we add a rule to deal with empty diffs:

$$\frac{\frac{\epsilon}{\quad} s}{\frac{\Delta}{\quad} s} = \frac{\Delta}{\quad} s$$

To demonstrate the utility of these ladders we derive two identities relating to empty diffs.

**Lemma 5.1.**  $(\Delta + \epsilon).\delta = \Delta.\delta$ .

$$\begin{array}{c}
\frac{\Delta.\delta}{s} \ast \Delta \quad \frac{\delta}{s} \quad \frac{(\Delta + \epsilon).\delta}{s \ast \Delta} \quad \frac{(\Delta + \epsilon).\delta}{s \ast (\Delta + \epsilon)} \quad \frac{\epsilon}{s} \quad \frac{(\Delta + \epsilon).\delta}{s \ast \Delta} \\
\frac{\Delta}{s} = \frac{\Delta + \epsilon}{s} = \frac{(\Delta + \epsilon).\delta}{s} = \frac{\Delta}{s} = \frac{\Delta}{s}
\end{array}$$

Figure 9:  $(\Delta + \epsilon).\delta = \Delta.\delta$

$$\begin{array}{c}
\frac{\Delta.\delta}{s} \ast \Delta \quad \frac{\delta}{s} \quad \frac{\delta + \epsilon}{s} \quad \frac{\Delta.(\delta + \epsilon)}{s \ast \Delta} \\
\frac{\Delta}{s} = \frac{\Delta}{s} = \frac{\Delta}{s} = \frac{\Delta}{s}
\end{array}$$

Figure 10:  $\Delta.(\delta + \epsilon) = \Delta.\delta$

*Proof.* See figure 9. □

**Lemma 5.2.**  $\Delta.(\delta + \epsilon) = \Delta.\delta$ .

*Proof.* See figure 10. □

We note that theorem 3.1 gives us  $\Delta \ast \Delta.\delta \ast \delta \ast \delta.\Delta$  in the case when  $|\Delta| = |\delta| = 1$ . Then, using only the first two rules, we can derive expressions for  $\Delta.\delta$  given that  $\Delta$  and  $\delta$  are of arbitrary length. In what follows we often break down  $\Delta$  and  $\delta$  into  $\Delta' + \Delta''$  and  $\delta' + \delta''$  respectively, and assume  $|\Delta'| = |\delta'| = 1$ .

$$\begin{array}{c}
\frac{\Delta.\delta}{s} \ast \Delta \quad \frac{\delta}{s} \quad \frac{\Delta'}{s} \quad \frac{\Delta''}{s} \quad \frac{\Delta''.\Delta'.\delta}{s \ast \Delta'} \quad \frac{\Delta''.\Delta'.\delta}{s \ast \Delta' \ast \Delta''} \\
\frac{\Delta}{s} = \frac{\Delta}{s} = \frac{\Delta'}{s} = \frac{\Delta'}{s} = \frac{\Delta}{s}
\end{array}$$

Figure 11:  $\Delta.\delta = \Delta''.\Delta'.\delta$  when  $\Delta = \Delta' + \Delta''$  and  $|\delta| = 1$

$$\begin{array}{c}
\frac{\Delta.\delta}{s} \ast \Delta \quad \frac{\delta}{s} \quad \frac{\delta'}{s} \quad \frac{\delta''}{s} \quad \frac{(\delta'.\Delta).\delta''}{s \ast \delta' \ast \delta''} \quad \frac{(\delta'.\Delta).\delta''}{s \ast \delta' \ast \delta''} \\
\frac{\Delta}{s} = \frac{\Delta}{s} = \frac{\Delta}{s} = \frac{\Delta}{s} = \frac{\Delta}{s}
\end{array}$$

Figure 12:  $\Delta.\delta = \Delta.\delta' + (\delta'.\Delta).\delta''$  when  $|\Delta| = 1$  and  $\delta = \delta' + \delta''$

**Lemma 5.3.**  $\Delta.\delta = \Delta''.\Delta'.\delta$  when  $\Delta = \Delta' + \Delta''$  and  $|\delta| = 1$ .

*Proof.* See figure 11. □

**Lemma 5.4.**  $\Delta.\delta = \Delta.\delta' + (\delta'.\Delta).\delta''$  when  $|\Delta| = 1$  and  $\delta = \delta' + \delta''$ .

*Proof.* See figure 12. □

Lemma 5.3 is equivalent to definition 4.4 and lemma 5.4, in the derivation of which theorem 3.1 was used, to lemma 4.1. We note in passing that  $|\Delta'.\delta|$  and  $|\delta'.\Delta|$  may not necessarily be 1, but this does not affect the veracity of the proofs.

In the case of  $\Delta$  and  $\delta$  both being of arbitrary length, the same approach can be used.

$$\begin{array}{c}
\frac{\Delta.\delta}{s} \quad s * \Delta \quad \frac{\delta}{s} \quad s \quad \frac{\Delta''}{s * \Delta'} \quad \frac{\Delta''}{s * \Delta'} \\
= \quad \frac{\Delta}{s} = \quad \frac{\Delta'}{s} = \quad \frac{\Delta'}{s}
\end{array}$$

$$\begin{array}{c}
\frac{((\Delta'.\delta').\Delta'').(\delta'.\Delta').\delta''}{s * \Delta' * \Delta'.\delta' * (\Delta'.\delta').\Delta''} \\
= \quad \frac{\Delta''.\Delta'.\delta'}{s * \Delta' * \Delta''} \\
= \quad \frac{\Delta''}{s * \Delta'} = \quad \frac{\Delta''.\Delta'.\delta' + ((\Delta'.\delta').\Delta'').(\delta'.\Delta').\delta''}{s * \Delta} \\
= \quad \frac{\Delta'}{s} = \quad \frac{\Delta}{s}
\end{array}$$

Figure 13:  $\Delta.\delta = \Delta''.\Delta'.\delta' + ((\Delta'.\delta').\Delta'').(\delta'.\Delta').\delta''$  when  $\Delta = \Delta' + \Delta''$  and  $\delta = \delta' + \delta''$

**Lemma 5.5.**  $\Delta.\delta = \Delta''.\Delta'.\delta' + ((\Delta'.\delta').\Delta'').(\delta'.\Delta').\delta''$  when  $\Delta = \Delta' + \Delta''$  and  $\delta = \delta' + \delta''$ .

*Proof.* See figure 13. □

This derivation relies on the identity  $\Delta'' + \Delta''.\Delta'.\delta' = \Delta'.\delta' + (\Delta'.\delta').\Delta''$ , which is proved next.

$$\begin{array}{c}
\frac{\Delta''.\Delta'.\delta'}{s * \Delta''} \quad \frac{\Delta'.\delta}{s} \quad \frac{\Delta''}{s} \\
\frac{\Delta'' + \Delta''.\Delta'.\delta'}{s} = \quad \frac{\Delta''}{s} = \quad \frac{\Delta''}{s} = \quad \frac{\Delta'.\delta}{s} \\
= \quad \frac{(\Delta'.\delta').\Delta''}{s * \Delta'.\delta'} \\
= \quad \frac{\Delta'.\delta}{s} = \quad \frac{\Delta'.\delta' + (\Delta'.\delta').\Delta''}{s}
\end{array}$$

Figure 14:  $\Delta'' + \Delta''.\Delta'.\delta' = \Delta'.\delta' + (\Delta'.\delta').\Delta''$

**Lemma 5.6.**  $\Delta'' + \Delta''.\Delta'.\delta' = \Delta'.\delta' + (\Delta'.\delta').\Delta''$ .

*Proof.* See figure 14. □

We can now prove the main theorem.

**Theorem 5.1.**  $*(\Delta' + \Delta'') * (\Delta' + \Delta'').(\delta' + \delta'') = *(\delta' + \delta'') * (\delta' + \delta'').(\Delta' + \Delta'')$ .

*Proof.* See figure 15, which shows just over the first half of the derivation. The second half is the same with  $\Delta$  and  $\delta$  interchanged. □

## 6 The general case

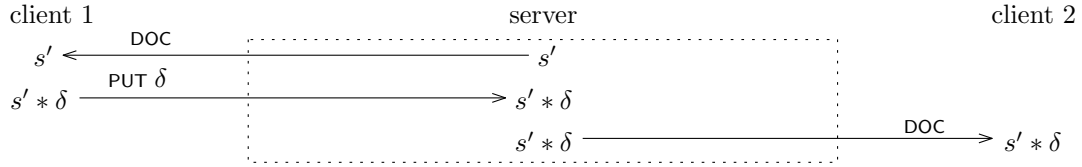
We have proved the identity  $*\Delta * \Delta.\delta = *\delta * \delta.\Delta$  where  $\Delta$  and  $\delta$  are arrays of diffs of arbitrary length. What we now prove is that if two clients have the same document initially, their documents will thereafter remain synchronised whenever neither have pending diffs on the server. In order to prove this, we make a reasonable assumption about any distributed application that employs this algorithm, namely that clients cannot put diffs on the server before they get the document. The proofs that follow are for two clients, but can easily be generalised to many.

$$\begin{aligned}
& \frac{(\Delta' + \Delta'') + (\Delta' + \Delta'').(\delta' + \delta'')}{s} = \frac{\Delta' + \Delta'' + \Delta''.\Delta'.\delta' + ((\Delta'.\delta').\Delta'').(\delta'.\Delta').\delta''}{s} \\
& \frac{((\Delta'.\delta').\Delta'').(\delta'.\Delta').\delta'' = s * \Delta' * \Delta'.\delta' * (\Delta'.\delta').\Delta''}{s * \Delta' * \Delta'' * \Delta''.\Delta'.\delta'} \\
& \frac{\Delta''.\Delta'.\delta'}{s * \Delta' * \Delta''} \\
& \frac{\Delta''}{s * \Delta'} \\
& = \frac{\Delta'}{s} \\
& \frac{(\delta'.\Delta').\delta'' = s * \delta' * \delta'.\Delta'}{s * \Delta' * \Delta'.\delta'} \quad \frac{\delta''}{s * \delta'} \quad \frac{\Delta''}{s * \Delta'} \\
& \frac{\delta'}{s} \quad \frac{\delta'}{s} \quad \frac{\Delta'}{s} \\
& \frac{\Delta''}{s * \Delta'} \quad \frac{\Delta''}{s * \Delta'} \quad \frac{\delta''}{s * \delta'} \\
& = \frac{\Delta'}{s} = \frac{\Delta'}{s} = \frac{\delta'}{s} = \dots
\end{aligned}$$

Figure 15:  $*(\Delta' + \Delta'') * (\Delta' + \Delta'').(\delta' + \delta'') = *(\delta' + \delta'') * (\delta' + \delta'').(\Delta' + \Delta'')$

**Lemma 6.1.** *Immediately both clients have the document, they and the server have identical copies.*

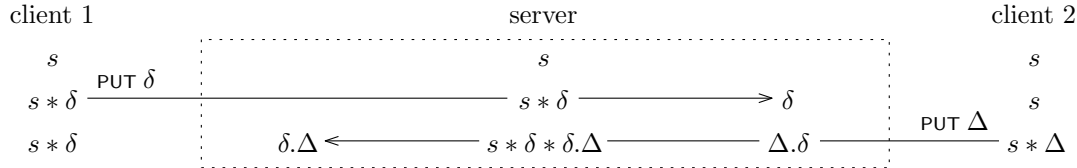
*Proof.* Since clients cannot put diffs on the server before they get the document, immediately both clients have the document, only one can subsequently have put diffs on the server and hence made amends to it. Without loss of generality we assume client 1 gets their document first:



Note that there are no pending diffs for client 2. The observation that the shared document is  $s' * \delta$  for both clients and the server completes the proof.  $\square$

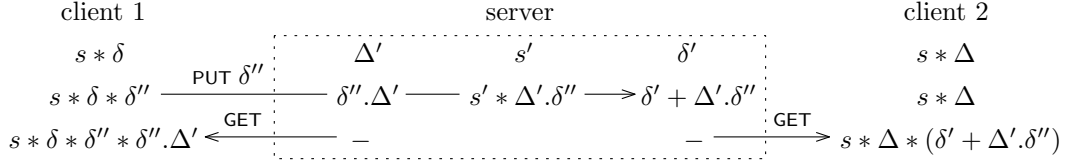
**Lemma 6.2.** *Suppose that two clients initially share a document with the server. If these clients subsequently put diffs on the server, which immediately amends them appropriately and applies them to its document, then should either client get their pending diffs and apply them to their own document, the resulting document is identical to the one held on the server.*

*Proof.* We use an inductive argument, the base case of which must include pending diffs for both clients. Up until this point it is straightforward to check that the copies of the documents remain identical. Without loss of generality we assume that client 1 puts their diffs on the server first:



Since  $s * \delta * \delta.Delta = s * \Delta * \Delta.Delta$  the base case is proved. We now set  $\Delta' = \delta.Delta$ ,  $\delta' = \Delta.Delta$  and  $s' = s * \delta * \delta.Delta$ . The induction step consists of one client putting further diffs on the server, with

the induction hypothesis being  $s * \delta * \Delta' = s' = s * \Delta * \delta'$ . We can safely assume this is client 1, since the case when client 2 puts further diffs on the server would be covered by the previous step:



The following equalities make use of the identity  $*\Delta * \Delta.\delta = *\delta * \delta.\Delta$  and the induction hypothesis:

$$\begin{aligned}
s * \delta * \delta'' * \delta''.\Delta' &= s * \delta * \Delta' * \Delta'.\delta'' \\
&= s' * \Delta'.\delta'' \\
&= s * \Delta * \delta' * \Delta'.\delta'' = s * \Delta * (\delta' + \Delta'.\delta'')
\end{aligned}$$

The observation that these equalities include the amended documents on either client should they get their pending diffs as well as the amended document on the server completes the proof.  $\square$

## 7 Conclusions

We have shown that it is possible by some jiggery-pokery to prove the identity  $*\Delta * \Delta.\delta = *\delta * \delta.\Delta$  in all the cases when  $\Delta$  and  $\delta$  are single diffs. Generalising the result to arrays of diffs proved difficult because of the unavoidable presence of double diffs. Specifically, an insert may split a delete. To work around this problem we came up with ladders, which stressed the relations between the various operators “.”, “\*” and “+” without getting bogged in the details. It is worth noting that the resulting derivations do not rely on an inductive argument. In fact we can show convincingly using this abstract approach that an inductive argument will never work. Suppose we want to prove the aforementioned identity when  $|\Delta| > 1$  and  $|\delta| = 1$ . If we proceed by induction on the length of  $\Delta$  we have the base case, and for the inductive step we set  $\Delta = \Delta' + \Delta''$  where  $|\Delta'| = 1$ . Our induction hypothesis is then  $*\Delta'' * \Delta''.\delta = *\delta * \delta.\Delta''$ . We now can only expand:

$$\begin{aligned}
*\Delta * \Delta.\delta &= *(\Delta' + \Delta'') * (\Delta' + \Delta'').\delta \\
&= *\Delta' * \Delta'' * \Delta''.\Delta'.\delta
\end{aligned}$$

But here we fail, as already pointed out. As  $|\Delta.\delta|$  may be 2, so we cannot use our induction hypothesis. Even worse,  $|(\Delta.\delta).\Delta''|$  may be nearly double the length of  $\Delta''$ . Inductive arguments will clearly never work. By contrast, the abstract approach does work, and suggests a branching algorithm that splits an array of diffs and deals with each sub-array separately. There is in fact no need, when splitting  $\Delta$  into  $\Delta'$  and  $\Delta''$ , to set  $|\Delta'| = 1$ . Is there some optimisation to be had from splitting  $\Delta$  halfway, perhaps? Finally, we conclude with the observation that it is easy to generalise this algorithm to more than two users by simply requiring that each user’s diff gets put on the stack of every other’s.

## References

- [1] Google docs  
<http://docs.google.com/>.
- [2] Keshav Kunal Sanjeev Khanna and Benjamin C. Pierce. A formal investigation of diff3  
<http://www.cis.upenn.edu/~bcpierce/papers/diff3-short.pdf>, 2012.